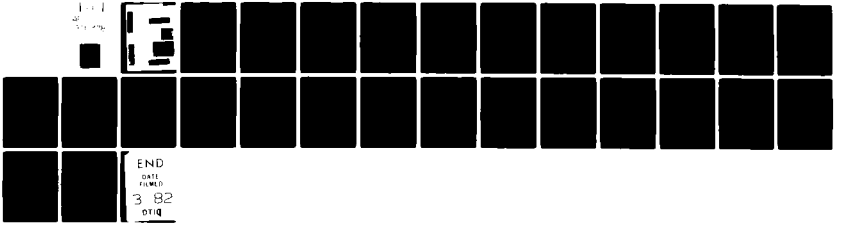
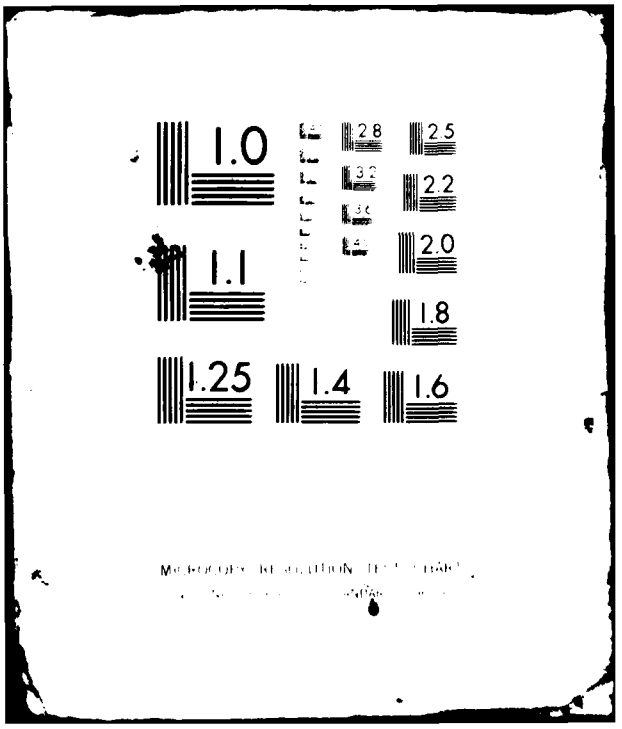
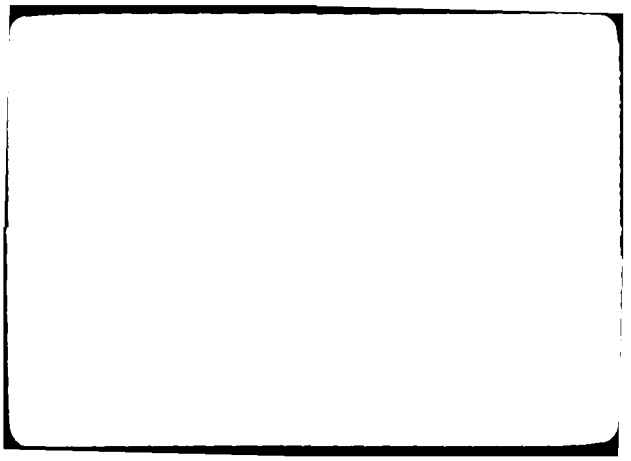
AD-A111 378 MINNESOTA UNIV MINNEAPOLIS DEPT OF COMPUTER SCIENCE F/G 12/2
A PARALLEL MATCHING ALGORITHM FOR CONVEX BIPARTITE GRAPHS AND A--ETC(U)
APR 81 E DEKEL, S SAHNI N00014-80-C-0650

UNCLASSIFIED TR-81-3 SBI-AD-E001 187 NL



END
DATE FILMED
3 82
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5



Computer Science Department
136 Lind Hall
Institute of Technology
University of Minnesota
Minneapolis, Minnesota 55455

A Parallel Matching Algorithm
for Convex Bipartite Graphs and
Applications to Scheduling

by

Eliezer Dekel and Sartaj Sahni

Technical Report 81-3
April 1981

Cover design courtesy of Ruth and Jay Leavitt.

A Parallel Matching Algorithm for Convex Bipartite Graphs and Applications to Scheduling*

Eliezer Dekel and Sartaj Sahni
University of Minnesota

Abstract

An efficient parallel algorithm to obtain maximum matchings in convex bipartite graphs is obtained. This algorithm can be used to obtain efficient parallel algorithms for several scheduling problems. Some examples are: job scheduling with release times and deadlines; scheduling to minimize maximum cost; and preemptive scheduling to minimize maximum completion time.

Key words and phrases: Parallel algorithm, convex bipartite graph, scheduling , complexity.

*This research was supported in part by the Office of Naval Research under contract N00014-80-C-0650.

1. Introduction

A convex bipartite graph G is a triple (A,B,E). A = $\{a_1,a_2,\ldots,a_n\}$ and B = $\{b_1,b_2,\ldots b_m\}$ are disjoint sets of vertices. E is the edge set. E satisfies the following properties:

(1) If (i,j) is an edge of E, then either i∈A and j∈B or i ∈B and j∈A; i.e., no edge joins two vertices in A or two in B.

(2) If $(a_i,b_j)\in E$ and $(a_i,b_{j+k})\in E$, then $(a_i,b_{j+q})\in E$, $1\le q<k$.

Property (1) above is the bipartite property while property (2) is the convexity property. An example convex bipartite graph is shown in Figure 1.1.

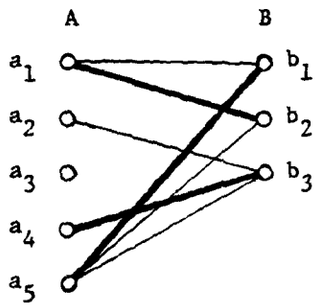


Figure 1.1 A convex bipartite graph.

$F \subseteq E$ is a matching in the convex bipartite graph $G=(A,B,E)$ iff no two edges in F have a common endpoint. $F1=\{(a_1,b_2), (a_4,b_3), (a_5,b_1)\}$ is a matching in the graph of Figure 1.1 while $F2=\{(a_1,b_1),(a_1,b_2),(a_2,b_3)\}$ is not. F is a maximum cardinality matching (or simply a maximum matching) in G iff (a) F is a matching and (b) G contains no matching H such that $|H|>|F|$ ($|H|$=number of edges in H). The matching depicted by solid lines in Figure 1.1 is a maximum matching in that graph.

In what follows, we shall find it convenient to have an alternate representation of convex bipartite graphs. It is clear that every convex bipartite graph G=(A,B,E), $A=\{a_1,\ldots,a_n\}$, $B=\{b_1,\ldots,b_m\}$ is uniquely represented by the set of triples:

$$T = \{(i,s_i,h_i)\mid 1\le i\le n\}$$
$$s_i=\min\{j\mid (a_i,b_j)\in E\}$$

$h_i = \max\{j \mid (a_i, b_j) \in E\}$

Intuitively, in the triple representation, we explicitly record the smallest (s_i) and highest (h_i) index vertices to which each a_i is connected. For the example of Figure 1.1, we have $T = \{(1,1,2), (2,3,3), (3,\emptyset,\emptyset), (4,3,3), (5,1,3)\}$.

As an example of the use of matchings in convex bipartite graphs, consider the problem of scheduling n unit time jobs to minimize the number of tardy jobs. In this problem, we are given a set, of n jobs. Job i has a release time r_i and a due time d_i. It requires one unit of processing. A subset F of J is a feasible subset iff every job in F can be scheduled on one machine in such a way that no job is scheduled before its release time or after its due time. A feasible subset F is a maximum feasible subset iff there is no feasible subset Q of J such that $|Q| > |J|$.

A maximum feasible subset F can be found by transforming the problem into a maximum matching problem on a convex bipartite graph. Without loss of generality, we may assume that all the r_is and d_is are natural numbers; $\min\{r_i\} = 0$; $r_i < d_i$, $1 \le i \le n$; and $\max\{d_i\} \le n$. The convex bipartite graph corresponding to J is given by the triple set $T = \{(i, s_i, h_i) \mid s_i = r_i, h_i = d_i - 1\}$. Figure 1.2 shows an example job set and the corresponding convex bipartite graph G. Vertex i of A represents job i while vertex i in B simply represents the time slot $[i, i+1]$. There is an edge from job i to time slot $[j, j+1]$ iff $r_i \le j < d_i$. Hence, every matching in G represents a feasible subset of J. Also, corresponding to every feasible subset of J there is a matching in G. Clearly, a maximum cardinality feasible subset of J can be easily obtained from a maximum matching of G. In addition, a maximum matching also provides the time slots in which the jobs should be scheduled.

We shall see several other examples of the application of matchings in convex bipartite graphs in later sections.

Glover [4] has obtained a rather simple algorithm to find a maximum matching in a convex bipartite graph $G = (A, B, E)$. Let $h_i = \max\{j \mid (a_i, b_j) \in E\}$, $1 \le i \le |A|$. Glover's algorithm considers the vertices in B one by one starting at b_1. We first determine the set R of remaining vertices in A to which the vertex b_j currently being considered is connected. Let q be such that $a_q \in R$ and $h_q = \min_{a_p \in R}\{h_p\}$. Vertex b_j is matched to a_q and a_q deleted from the graph. The next vertex in B is now considered. Figure 1.3 specifies Glover's algorithm more formally. By using appropriate data structures, Glover's algorithm can be implemented to run in $O(mn)$ time.

$r_i=s_i$	$d_i=h_i+1$
0	2
0	3
2	4
2	6
4	5
2	6

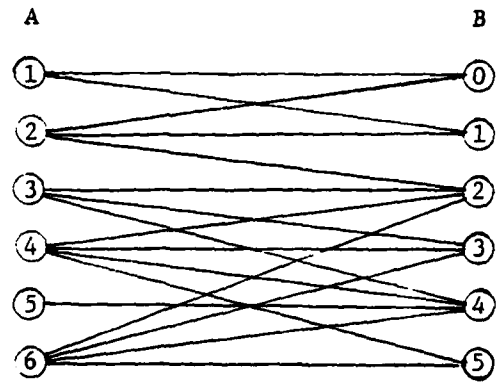


Figure 1.2

```
procedure MATCH(G,m,n,h)
   ///G=(A,B,E) is a convex bipartite graph.|A|=n,|B|=m//
   //n_j=max{j|(a_i,b_j)∈E}.A maximum matching//
   //M^1 is found//
   set M; integer m,n,n_{1:n} graph G
   M ← ∅
   for i ← 1 to m do
      let R be the set of remaining vertices to which b_i
      is connected
      if R≠∅ then q ← j such that j∈R and n_j = min {h_p}
                                                p∈R
             M ← M U{(a_q,b_i)}
             delete a_q from G
             endif
   end for
   return ()
end MATCH
```

Figure 1.3 Glover's maximum matching algorithm.

In this paper, we are primarily concerned with parallel algorithms. In Section 2, we obtain a parallel algorithm for maximum matchings in convex bipartite graphs. In Section 3, we use this parallel algorithm to obtain efficient parallel algorithms for three scheduling problems.

The parallel computer model used is the shared memory model (SMM). This is an example of a single instruction stream multiple data stream (SIMD) computer. In a SMM computer, there are p processing elements (PEs). Each PE is capable of performing the standard arithmetic and logical

operations. The PEs are indexed $0,1,\ldots,p-1$ and an individual PE may be referenced as in PE(i). Each PE knows its index and has some local memory. In addition, there is a global memory to which every PE has access. The PEs are synchronized and operate under the control of a single instruction stream. An enable/disable mask may be used to select a subset of the PEs that are to perform an instruction. Only the enabled PEs will perform the instruction. Disabled PEs remain idle. All enabled PEs execute the same instruction. The set of enabled PEs may change from instruction to instruction.

If two PEs attempt to simultaneously read the same word of the shared memory, a read conflict occurs. If two PEs attempt to simultaneously write into the same word of the shared memory, a write conflict occurs. Throughout this paper, we shall assume that read and write conflicts are prohibited.

The reader is referred to [2] for a list of references dealing with graph algorithms, matrix algorithms, sorting, scheduling, etc. on a SMM computer.

2. Parallel Matching In Convex Bipartite Graphs

In Section 1, we showed that every instance of the problem of scheduling jobs to minimize the number of tardy jobs could be transformed into an equivalent instance of the maximum matching in a convex bipartite graph problem. It should be evident that the reverse is also true. Hence, the two problems are isomorphic. A parallel algorithm for a special case of the job scheduling formulation was obtained by us in [1]. In this special case, it was assumed that all jobs have the same release time. This corresponds to the case when the convex bipartite graphs are of the form $T=\{(i,s_i,h_i) \mid 1 \le i \le n\}$ $s_i=c, 1 \le i \le n$ for some c.

We shall now proceed to show how the solution for the special case described above can be used to solve the general case when all the r_is are not the same. This will be done using the binary tree method described by Dekel and Sahni [2]. Rather than specify the new algorithm formally, we shall describe how it works by means of an example.

A convex bipartite graph is shown in Figure 2.1. For this graph, $|A|=14$ and $|B|=13$. The s_i and h_i values associated with each vertex of A are given in the first two columns of this figure. The first step in our parallel algorithm for maximum matching is to sort the vertices in A in nondecreasing order of s_i. Vertices with the same s_i are sorted into nondecreasing order of h_i. For our example, the

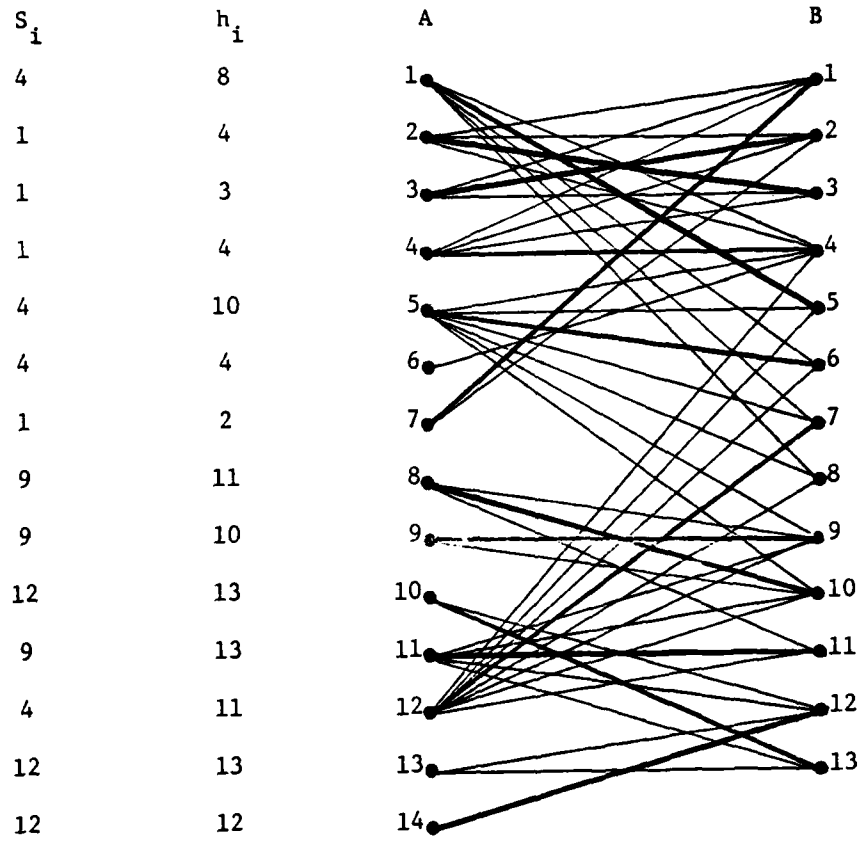


Figure 2.1

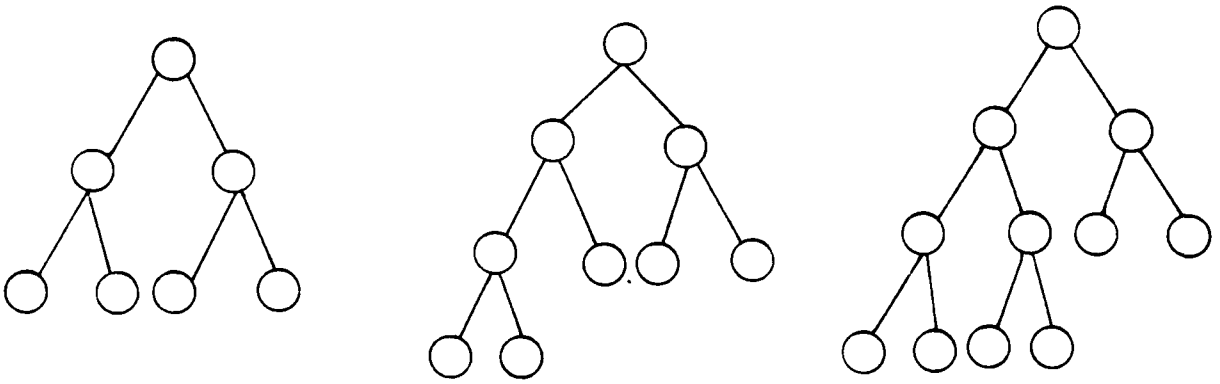
result of this reordering is shown in Figure 2.2.

i	7	3	2	4	6	1	5	12	9	8	11	14	10	13
S_i	1	1	1	1	4	4	4	4	9	9	9	12	12	12
h_i	2	3	4	4	4	8	10	11	10	11	13	12	13	13

Figure 2.2

Following the sort, we identify the distinct s_i values. Let these be $R_1, R_2, \ldots, R_k$. Assume that $R_1 < R_2 < \ldots < R_k$. Let $R_{k+1} = \max\{d_i\}+1$. For our example, k=4 and $R(1:k+1) = (1, 4, 9, 12, 14)$.

We are now ready to use the binary tree method of [2]. The underlying computation tree we shall use is the unique complete binary tree with k leaf nodes. Figure 2.3 shows the complete binary trees with 4, 5, and 6 leaf nodes. For our example, k=4 and we use the tree of Figure 2.3(a). With each node, P, in the computation tree, we associate a contiguous subset $\{u, u+1, u+2, \ldots, v\}$ of the vertices in B. This subset is denoted [u,v].P or simply [u,v].



(a) 4 leaves (b) 5 leaves (c) 6 leaves

Figure 2.3: Complete binary trees.

Let the leaf nodes of the computation tree be numbered 1 through k, left to right. If P is the ith leaf node, then [u,v].P is $[R_i, R_{i+1}-1]$ (i.e., $u=R_i$ and $v=R_{i+1}-1$). If P is not a leaf node, then the subset of B associated with P is [u,v].LC(P) U [u,v].RC(P) where LC(P) and RC(P) are, respectively, the left and right children of P. The subsets of B associated with each of the vertices in the computation tree for our example are shown in Figure 2.4. The number in each node of this tree is its index.

Let P be any vertex of the computation tree. Let [u,v] be the subset of B associated with P. The subset of A *available for matching* at node P is denoted M(P) and is defined to be:

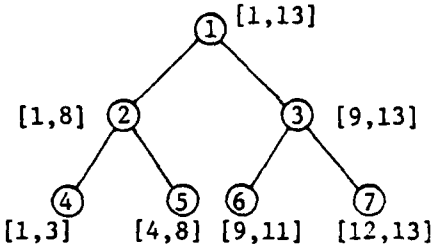


Figure 2.4

$$M(P)=\{i \mid u \leq s_i \leq v\}$$

For, example,

M(1)={1,2,...,14};
M(2)={1,2,3,4,5,6,7,12};
M(4)={2,3,4,7};
etc.

The subset M(P) of A vertices available for matching at P may be partitioned into three subsets Q(P), I(P), and T(P). Q(P) is a maximum cardinality subset of M(P) that may be matched with vertices in [u,v].P by algorithm MATCH; this subset is called the matched set. I(P) denotes the infeasible set. It consists of all vertices $i \in M(P)-Q(P)$ such that $h_i \leq v$. The transferred set T(P) consists of all vertices $i \in M(P)-Q(P)$ such that $h_i > v$.

Consider node 2 of Figure 2.4. The matching problem defined at this node is given in Figure 2.5. If Glover's algorithm is used on this graph, then {1,2,3,4,5,7,12} defines a subset of A' that can be matched with vertices in B'. Further, this gives a maximum matching. Hence, Q(2)={1,2,3,4,5,7,12}; I(2)=6; and T(2)=ϕ. Observe that |Q(1)| is the size of a maximum matching in the original convex bipartite graph. Also, T(1)=ϕ and I(1)=A-Q(1).

We shall make two passes over the computation tree. The first pass begins at the leaves and moves towards the root. During this pass, the Q, I, and T sets for each node are computed. The second pass starts at the root and progresses towards the leaves. In this pass, the Q set for each node is updated so as to correspond to the set of A vertices matched by Glover's algorithm to the B vertices associated with the node.

s_i	h'_i	A'	B'
4	8	1 ○	○ 1
1	4	2 ○	○ 2
1	3	3 ○	○ 3
1	4	4 ○	○ 4
4	8	5 ○	○ 5
4	4	6 ○	○ 6
1	2	7 ○	○ 7
4	8	12 ○	○ 8

Figure 2.5

Pass 1

In this pass, we make extensive use of the parallel algorithm developed in [1] for the case when all the s_is are the same. For our purposes here, it is sufficient to know the sequential algorithm (FEAS of [1]) that this parallel algorithm is based on. This sequential algorithm is given in Figure 2.6. For convenience, this has been translated into the graph language used here.

```
line procedure FEAS(n,u,v)
       //find a maximum matching of vertices in A onto the B
       set [u,v] for every vertex i∈A, s_i=u//
  1    global MAT(1:n); set A; integer n,u,v,i,j
  2    sort A into nondecreasing order of h_i
  3    MAT(1:n) ← 0//initialize//
  4    j ← u
  5    for i ← 1 to n do
  6      case
  7      :j>v: return(j) //all vertices in B matched//
  8      :j<h_i: //select i// j ← j+1, MAT(i) ← 1
  9      end case
 10    end for
 11    return(j)
 12  end FEAS
```

Figure 2.6

An examination of Glover's algorithm (Figure 1.3) reveals that it performs exactly as does procedure FEAS when the restrictions and simplifications applicable to FEAS are incorporated into Figure 1.3.

Hence, for a leaf node of the computation tree, the Q set may be obtained by a direct application of procedure FEAS (or its parallel equivalent). For example, for node 4 of Figure 2.4, we have A = M(4) = {2,3,4,7}; h_2=4; h_3=3; h_1=4; h_7=2; u=1; and v=4. Using FEAS, we obtain Q(4)={7,3,2}. Note that Q consists of exactly those vertices i with MAT(i)=1. I() consists of exactly those vertices i with MAT(i)=ϕ and $h_i \leq v$. The remaining vertices form T(). The matched set Q, transferred set T, and infeasible set I for each of the leaves in our example are shown in Figure 2.6. Null sets are not explicitly shown. So, for node 4, I(4)=ϕ; T(4)={4}; and Q(4)={7,3,2}. The sets are ordered by h_i.

For a nonleaf node P, the Q, I, and T sets may be obtained by using the Q, I, and T sets of the children of P. Let L and R, respectively, be the left and right children of P. To determine Q(P), we use procedure FEAS with u=u_R and v=v_R ($[u_R,v_R]$ is the subset of B associated with the right child R of P). The A set consists of T(L) U Q(R). Since both T(L) and Q(R) are already sorted by h_i, the sort of line 2 of FEAS can be replaced by a merge.

Let S be the subset of T(L) U Q(R) that has MAT()=1 following the execution of FEAS. The following theorem establishes that Q(L) U S is a maximum cardinality subset of M(P) that may be matched with vertices in [u,v].P. Hence, Q(P)=Q(L) U S.

Theorem 2.1: S as defined above is a maximum cardinality subset of M(P) that may be matched with vertices in [u,v].P using algorithm MATCH.

Proof: The proof is by induction on the distance of P from the farthest leaves in the subtree of which it is a root. If this distance is 1, then Q(L) and Q(R) are maximum cardinality subsets of M(L) and M(R) that can, respectively, be matched by Glover's algorithm with vertices in [u,v].L and [u,v].R. If this distance is greater than 1, then Q(L) and Q(R) satisfy this maximum cardinality matching requirement by induction.

As far as node P is itself concerned, we see that only vertices in M(L) are candidates for matching with vertices in $[u_L,v_L]$ (recall that for vertices in M(R), the s_i value exceeds v_L). Furthermore, when Glover's algorithm is used with the A set being M(P) and the B set being $[u_L,v_R]$ = [u,v].P, vertices in B are considered in the order u_L, $u_L+1,\ldots,v_L$, $u_R,\ldots,v_R$. Hence, Q(L) is precisely the subset

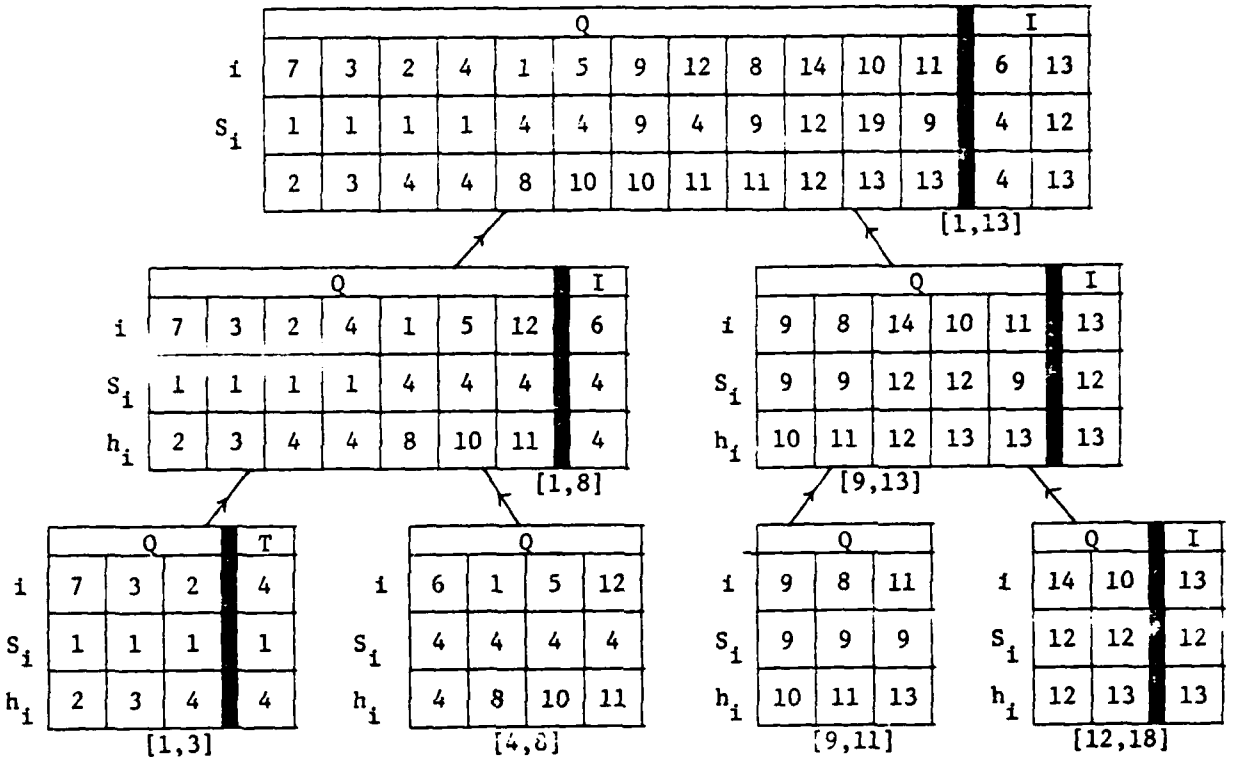


Figure 2.7: *Results of first pass.*

of M(P) that gets matched with vertices in $[u_L, v_L]$.

The candidates for the remaining vertices in B, i.e., $[u_R, v_R]$ are clearly T(L) U M(R). From the way Glover's algorithm works, it is also clear that the vertices of T(L) U M(R) that will get matched to $[u_R, v_R]$ are a subset of T(L) U Q(R). Let this subset be S'. We wish to show that S is a legitimate choice for S'. First, we show that S represents a feasible matching. Then, we shall show that S is in fact selectable by Glover's algorithm.

We know that Q(R) can be matched into $[u_R, v_R]$. Let Z be any such matching. Since S is selected by FEAS, we know that every vertex in S can be matched to a vertex in $[u_R, v_R]$ in a such a way that no vertex j in S is matched to a vertex with index greater than h_j. Consider a matching W that

meets this condition. Now suppose that some vertex j in S is matched to a vertex q in $[u_R,v_R]$ with index less than s_j. Clearly, j must be a member of Q(R) (as all vertices in T(L) have an s value less than u_R). Suppose that j is matched to j_1 in Z. If j_1, is free in W, then the matching of j in W may be changed from q to j_1. If j_1 is not free, then suppose it is matched to j_2. If $j_2 \in T(L)$, then j_2 may be matched to q and j to j_1 (note that $q<j_1$ and so $s_{j_2}<q<h_{j_2}$). If $j_2 \in Q(R)$, then suppose that j_2 is matched to j_3 in Z. It is easy to see that $j_3 \neq j_1$. If $q=j_3$ or j_3 is free in W, then we may match j to j_1 and j_2 to q. Otherwise, let j_4 be the vertex matched to j_3 in W. It should be clear that we can continue in this way and modify W so that j is matched to j_1, j_2 to j_3, j_4 to j_5, etc. In the new matching, there is one fewer vertex of S that is matched to a vertex with smaller s value.

Repeating the above construction several times, W can be transformed into a matching such that every vertex $j \in S$ is matched to a vertex q such that $u_R \leq s_j < q < h_j \leq v_R$. Hence, S represents a feasible matching.

Let S' (as defined earlier) be the subset of Q(R) U T(L) matched by Glover's algorithm to the vertex set $[u_R,v_R]$. We shall now proceed to show that S is a valid choice for S'. Let Z be any matching of Q(R) into $[u_R,v_R]$. Let Y be a matching of S' in which all vertices in $Q(R) \cap S'$ are matched to the same vertex in $[u_R,v_R]$ as in the matching Z. Let W be a corresponding matching for S. The existence of the matchings Y and W is a consequence of the construction used to show the feasibility of S.

From the definition of S', it follows that $S' \not\subset S$. Also, from the working of FEAS, it follows that $S \not\subset S'$. Let $j \in S$ be a vertex with least h_j such that $j \notin S'$. If no such j exists, then S=S'. Assume that j is matched to q in W. If q is free in Y, then S' cannot be of maximum cardinality. So, let $p \in S'$ be matched to q in Y. By definition of Y and W, $p \notin S$. Also, from the order in which FEAS considers vertices, $h_p > h_j$. Hence, $S' \cup \{j\} - \{p\}$ is also a subset selectable by Glover's algorithm. $S' \cup \{j\} - \{p\}$ agrees with S in one place more than does S'.

By repeating this interchange process, S' may be transformed into S with the result that S is also a maximum cardinality subset of Q(R) U T(L) that is selectable by Glover's algorithm for matching in $[u_R,v_R]$.

Hence, Q(L) U S is a maximum cardinality subset of M(P) selectable by Glover's algorithm for matching in [u,v].P. []

Once Q(P) is known, T(P) and I(P) are easily computed. Actually, as I(P) is never used, we may omit its computation. Figure 2.6 shows the Q, I, and T sets (except when empty) for all nodes in our example.

Pass 2

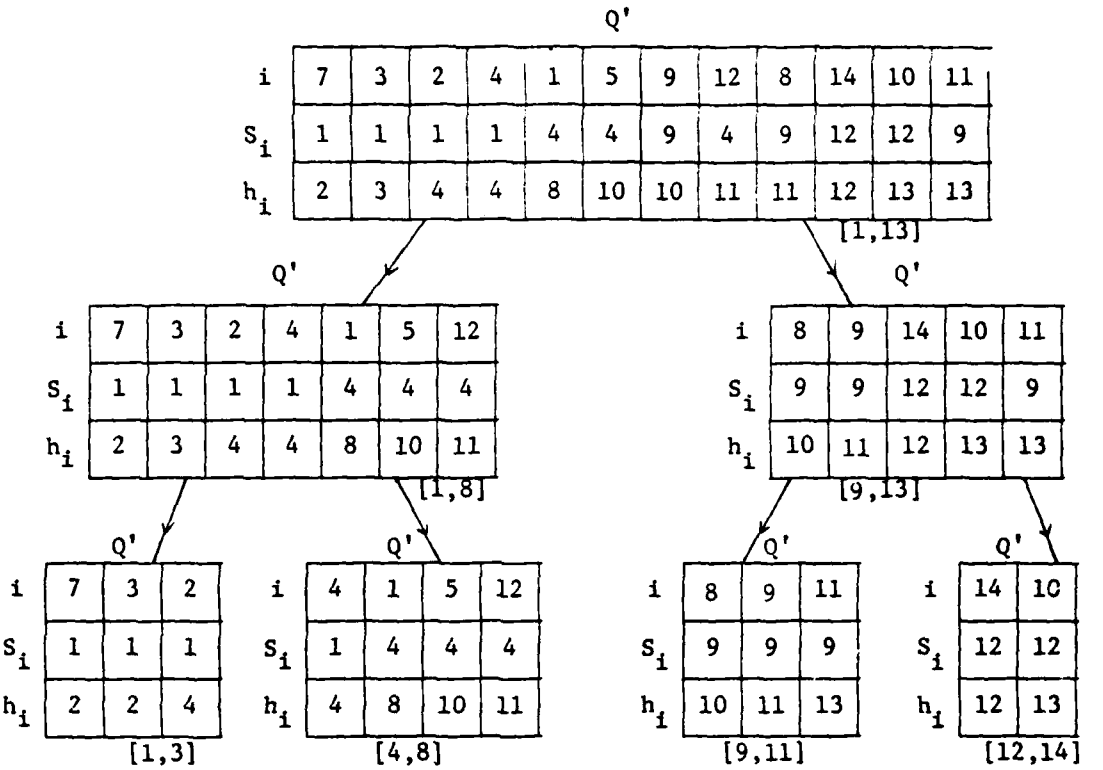
In the second pass, for each vertex P of the computation tree, we compute a set Q'(P) which represents the set of A vertices matched by Glover's algorithm with the set [u,v].P. With repsect to the matching shown by solid lines in Figure 2.1, we see that if P is the root, then Q, (P) = {1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 14}; if P is node 3 of the computation tree, then Q'(P) = {8, 9, 10, 11, 14}.

If P is the root node, then Q'(P)=Q(P), by definition of Q(P). Let P be any nonleaf node for which Q'(P) has been computed. Let L and R be the left and right children, respectively, of P. Let $[u,v].L = [u_L,v_L]$ and $[u,v].R = [u_R,v_R]$. Let $V = \{j \mid j \in Q'(P) \text{ and } h_j < u_L\}$. Let W be the ordered set obtained by merging together V and Q(L) (note that both V and Q(L) can be maintained so that they are in nondecreasing order of h_i and that W is also in nondecreasing order of h_i). Q'(L) consists of the first min $\{|W|, v_L-u_L+1\}$ vertices in W. The correctness of this statement may be established by induction on the distance of P from the root. Q'(R) is readily seen to be Q'(P)-Q'(L). Figure 2.7, shows the Q'( ) sets for all the vertices in the computation tree of our example.

From the Q'( ) sets of the leaves, the matching is easily obtained. If P is a leaf, and [u,v].P=[a,b], then the first vertex in Q'(P) is matched with a, the second with a+1, etc. (note that Q'(P) is in nondecreasing order of h_i). The matching for our example is also given in Figure 2.7.

Complexity Analysis

The initial ordering of A by h_i and within h_i by S_i can be done in $O(\log^2 n)$ time using n/2 PEs([6] and [7]). During the first pass, the computation of Q() requires the use of FEAS and a merge. The use of FEAS (without the sort) takes O(logn) time and requires $O(|M(P)|/\log|M(P)|)$ PEs. The merge at node P takes O(logn) time with |M(P)|/2 PEs. Since, Q can be computed in parallel for all nodes on the same level of the computation tree, O(logn) time is needed per level. The total time for the first pass is $O(\log^2 n)$ and n/2 PEs are needed. Pass 2 requires only some merging per node. The total cost of this pass is also $O(\log^2 n)$ and n/2 PEs suffice.



matching={(7,1),(3,2),(2,3),(4,4),(1,5),(5,6),(12,7),(9,9),(8,10)
(11,11),(14,12),(10,13)}

Figure 2.8 Second pass

Hence, the overall complexity of our parallel algorithm for maximum matching in convex bipartite graphs is $O(\log^2 n)$. The PE requirement is $O(n)$.

Another complexity measure often computed for parallel algorithms is the effectiveness of processor utilization (EPU) (see [1], [2], and [9]). For any problem P and parallel algorithm A, this is defined as follows:

EPU(P,A) =

$$\frac{\text{complexity of fastest sequential algorithm for P}}{\text{complexity of A} * \text{number of PEs used by A}}$$

For our algorithm, we have an EPU that is $O(nm/(\log^2 n * n)) = O(m/\log^2 n)$ (recall that $m=|B|$).

3. Applications to Scheduling

We have already seen one application of matchings in convex bipartite graphs to scheduling. This application was to a problem that is actually isomorphic to the matching problem; viz. Find the maximum number of unit time jobs from $J=\{(r_i,d_i) \mid 1\leq i\leq n\}$ that can be scheduled on a single machine such that no job is scheduled before its release time r_i or after its due time d_i. In this section, we shall look at three other scheduling problems that can be solved in an efficient manner using the parallel matching algorithm developed in previous section.

3.1 Scheduling to Minimize the Maximum Cost

In this problem, we are given n unit time jobs. Associated with each job is a release time r_i and a nondecreasing cost function $c_i()$, $1\leq i\leq n$. Let $S=(s_1,s_2,\ldots,s_n)$ be a one machine schedule for the n jobs. s_i denotes the start time of job i. Let q_i be as defined below:

$$q_i=\begin{cases} c_i(s_i) & s_i\geq r_i \\ oo & s_i<r_i \end{cases}$$

The cost of S is defined to be $\max\{q_i\}$. We wish to find a schedule with minimum cost.

As pointed out by Rinnooy Kan [8], this problem may be solved by first determining the finish time of a minimum finish time schedule for the n jobs. If the n jobs are already in nondecreasing order of release times r_i, then a minimum finish time schedule has start times s_i given by:

$$(3.1) \qquad s_1=r_1$$
$$s_i=\min\{r_i,\ s_{i-1}+1\}$$

This equation is a special case of Equation (6.1) of [1] and so we may use algorithm PADMIS of [1] to obtain the s_is. The finish time, C*, of a minimum finish time schedule for the n jobs is $s_n + 1$.

For simplicity, let us assume that $r_1 = \emptyset$. Once C* has been computed, a schedule that has minimum cost may be obtained by constructing the convex bipartite graph, G, with $A=\{1,2,\ldots,n\}$ and $B=\{\emptyset,1,\ldots,C^*-1\}$. There is an edge from i in A to j in B iff $j\geq r_i$. (A represents the jobs and j $\leftrightarrow$ B represents the time slot $[j,j+1]$). A cost $c_i(j)$ is associated with edge (i,j).

From our choice of C*, it follows that the graph just constructed has a maximum matching M of size n. Every such matching defines a feasible schedule. A matching for which the maximum edge cost is minimum defines the optimal matching for our scheduling problem.

As suggested by Rinnooy Kan [8], this matching can be obtained by performing a binary search on the $O(n^2)$ edge costs in G. At each iteration of this binary search, all edges with cost greater than c are deleted from G and a maximum matching in the resulting convex bipartite graph G' found. If this matching is of size less than n, then there is no matching in G with maximum edge cost no more than c; otherwise there is.

Complexity Analysis

The initial sort by release times can be done in $O(\log^2 n)$ time using $O(n)$ PEs [7]. C* can be found in $O(\log n)$ time using n PEs and the algorithm of [1]. Each iteration of the binary search requires the construction of G' and the solution of a matching problem. The first of these can be done in $O(\log^2 n)$ time using $O(n^2/\log^2 n)$ PEs. The second task takes $O(\log^2 n)$ time and $O(n)$ PEs. The total time needed to determine the matching with minimum cost is therefore $O(\log^3 n)$ and the number of PEs needed is $O(n^2/\log^2 n)$. Hence, the overall complexity of the parallel scheduling algorithm is $O(\log^3 n)$. The number of PEs used is $O(n^2/\log^2 n)$. The complexity of the fastest sequential algorithm known for this problem (i.e., the algorithm described above) is $O(n^2 \log n)$. Hence, the EPU of our parallel algorithm is $O(n^2 \log n/(\log^3 n * n^2/\log^2 n))=O(1)$.

3.2 Job Sequencing With Release Times and Deadlines

Once again, each job has a processing requirement that is one unit. However, this time there is a release time r_i, deadline d_i, and weight w_i associated with job i. No job may be scheduled before its release time. We are interested in obtaining a maximum weight subset of the n jobs such that all jobs in this subset can be scheduled on a single machine in such a manner that none starts before its release time or finishes after its deadline.

This problem can be solved efficiently by a sequential greedy algorithm. Jobs are considered in nondecreasing order of the weights. If the job, i, currently being considered can be scheduled together with all the others so far scheduled such that no job violates its release time or deadline, then job i is in the optimal schedule. Otherwise, it is not. This feasibility test can be carried out in O(n)

time. So, the sequential algorithm has complexity $O(n^2)$.

For the parallel algorithm, we resort to a convex bipartite graph formulation. The A set represents the n jobs; so, $|A|=n$. The B set represents time slots. Let $a=\min\{r_i\}$ and $b=\max\{d_i\}$. $B = \{a,a+1,\ldots,b-1\}$. In the convex bipartite graph corresponding to the scheduling problem, there is an edge from i in A to $j \in B$ iff $r_i \leq j < d_i$. The weight of each edge incident on vertex $i \in A$ is w_i. We wish to find a matching M for which the sum of edge weights is maximum.

Our parallel algorithm for this is based on the greedy sequential algorithm for job sequencing with release times and deadlines. Define the binary relation R on the weights w_i as follows:

$$w_i \; R \; w_j \text{ iff } w_i < w_j \text{ or } (w_i = w_j \text{ and } i > j)$$

Let G_i be the convex bipartite graph obtained from G by deleting all edges with weight w_j such that $w_j \; R \; w_i$. Clearly, the following determines whether or not i is in the maximum weight matching:

a) Determine the size of the maximum cardinality matching in G_i. Let this be s_i.
b) Delete all edges incident on vertex $i \in A$ of G_i. Determine the size of the maximum cardinality matching in the new graph. Let this be s'_i.
c) i is in the maximum weight matching iff $s_i = s'_i + 1$.

So, each vertex can determine, in parallel, whether or not it is in the maximum weight matching. To avoid read conflicts, we need to make n copies of the graph G. This can be done in $O(\log n)$ time using $O(n^2)$ PEs. G_i can be constructed in $O(1)$ time using $O(n^2)$ PEs per vertex i. Then, steps a), b), and c) above can be performed in $O(\log^2 n)$ time using a total of $O(n^2)$ PEs. (Note that only pass 1 of the parallel convex matching algorithm need be executed.) The total time needed to determine the subset of A in the maximum weight matching is therefore $O(\log^2 n)$. The number of PEs can easily be reduced to $O(n^2/\log n)$.

Once we have obtained the above subset of A, the actual matching can be determined by deleting from G all vertices that are not in this subset. A maximum cardinality matching on the resulting graph yields the desired matching with maximum weight. This corresponds to an optimal schedule for the scheduling problem.

The complexity of the parallel algorithm is $O(\log^2 n)$ and its EPU is $O(n^2/(\log^2 n * n^2/\log n)) = O(1/\log n)$.

Exmaple 3.1: Figure 3.1 gives an example job set. Let

$A_i=\{j|iRj\}$. Figure 3.2 gives the H_is for our example. The convex graph G_4 is shown in Figure 3.3. The maximum matching in G_4 is shown by broken lines. The maximum matching in the graph obtained from G_4 by deleting edges incident on vertex 4 is shown by wavy lines. These two matchings are of the same cardinality. Hence, vertex 4 is not in the maximum weight matching. []

i	1	2	3	4	5	6	7	8	9	10	11
r_i	1	4	5	1	3	0	4	2	6	2	4
d_i	3	6	6	3	6	1	6	6	7	3	7
w_i	50	55	65	40	70	20	60	80	60	30	85

Figure 3.1

3.3 Preemptive Scheduling With Precedence Contraints

Let J be a set of n unit time jobs. Let P be a precedence relation on J. For i∈J and j∈J, iPj iff i must be completed before j can commence. We may assume that P is a partial order. Hence, P may be represented as a directed acyclic graph (dag) as in Figure 3.4 (strictly speaking, P is the transitive closure of this graph). The directed edge <i,j> means that i must be completed before j can start (i.e., iPj). We also assume that jobs have been indexed so that iPj implies i<j. This is true of the indexing (of nodes) in Figure 3.4.

Muntz and Coffman [5] have developed a level algorithm to obtain minimum finish time preemptive schedules for J (as above) when the number of machines available is 2. Their algorithm also works for the case when the processing times are mutually commensurable (rather than simply 1 unit). A set of times $\{t_1, t_2, \ldots, t_n\}$ is mutually commensurable iff each is a multiple of some number w. In the case of mutually commensurable times, each job is broken into a chain of jobs each requiring w units of processing. In this section, we shall deal directly only with the case of unit time tasks.

H_1 = {2,3,5,7,8,9,11}
$s_i = 6$ $s_i' = 5$ Job 1 is in

H_2 = {3,5,7,8,9,11}
$s_i = 5$ $s_i' = 5$ Job 2 is out

H_3 = {5,8,11}
$s_i = 4$ $s_i' = 3$ Job 3 is in

H_4 = {1,2,3,5,7,8,9,11}
$s_i = 6$ $s_i' = 6$ Job 4 is out

H_5 = {8,11}
$s_i = 3$ $s_i' = 2$ Job 5 is in

H_6 = {1,2,3,4,5,7,8,9,10,11}
$s_i = 7$ $s_i' = 6$ Job 6 is in

H_7 = {3,5,8,11}
$s_i = 5$ $s_i' = 4$ Job 7 is in

H_8 = {11}
$s_i = 2$ $s_i' = 1$ Job 8 is in

H_9 = {3,5,7,8,11}
$s_i = 5$ $s_i' = 5$ Job 9 is out

H_{10} = {1,2,3,4,5,7,8,9,11}
$s_i = 6$ $s_i' = 6$ Job 10 is out

H_{11} = {Ø}
$s_i = 1$ $s_i' = 0$ Job 11 is in

Figure 3.2

Let G be the dag representation of the precedence relation P. A node in G is terminal iff its out-degree is Ø. The level of a node in G is the length of the longest path from v to any terminal node. All terminal nodes have level Ø. In Figure 3.4, the number outside each node is its level.

Muntz and Coffman's [5] algorithm to obtain minimum finish time 2-machine schedules is given in Figure 3.5. One observes that the objective of the for loop of this algorithm is to minimize the number of sets S_i with $|S_i|=1$. This can be done in parallel as follows. Let L be the maximum level in the precedence graph G. Let L_i be the level of vertex i in G. Let D_i be the length of the longest path from any vertex with no predecessors to i. Let $H_i=L-D_i$. The $[L_i,D_i]$ pairs of each vertex in our example is given in Figure 3.4. Note that H_i is the largest j such that job i could possibly be in S_j following the execution of the for loop in the Muntz-Coffman algorithm.

A job i is said to be critical iff $L_i=H_i$. Only non-critical jobs are candidates for displacement to other sets. Also, only those sets S_i that initially have exactly one critical job are candidates to receive a new job during the for loop. Note that every S_i must have at least one

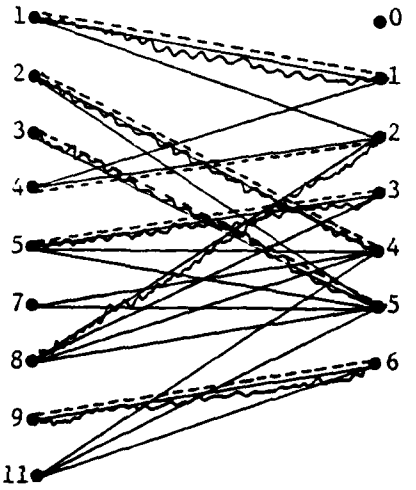


Figure 3.3

critical job. In Figure 3.6, the noncritical jobs have been marked with an X. Sets that are candidates to receive a new job have also been marked with an X. Double vertical lines mark set boundaries.

To determine the movement of noncritical jobs to candidate sets, we construct a convex bipartite graph in which the A set consists of the noncritical jobs while the B set consists of the candidate sets. There is an edge from $i \in A$ to $j \in B$ iff $L_i \leq j \leq d_i$. For our example, the resulting graph is shown in Figure 3.7.

We intend to use a maximum matching M in the graph constructed above as follows: if (i,j) is an edge of the matching M, then job i is moved to set S_j; vertices of A that are not matched to any vertex in B remain in their original set.

The maximum matching M is to be obtained from the convex bipartite graph constructed above as follows:

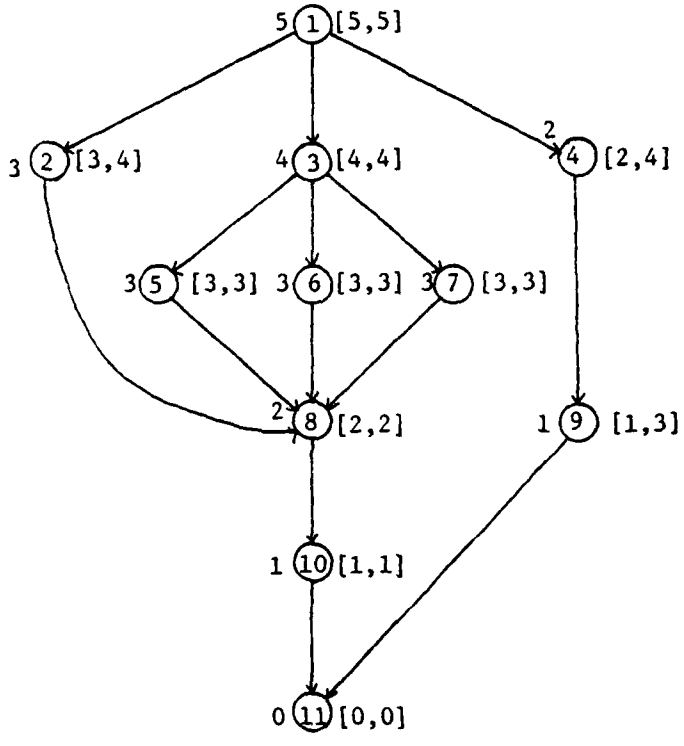


Figure 3.4

```
procedure SCH
   L ← maximum level
   S_i ← all jobs in J with level=i, 0≤i≤L
   for k ← L to 1 do
      if |S_k|=1 then find the largest j such that j<k and
                      S_j contains a job all of whose pro-
                      cessors are in S_L, S_L-1,..., S_k-1.
                      If no such j exists then exit endif.
                      Let q be this value of j and let r
                      be the job in S_q with the above pro-
                      perty
                       S_q ← S_q - {r}; S_k ← S_k U {r}
      endif
   endfor
   preemptively schedule the jobs in each S_i using
   McNaughton's rule.
   Jobs in S_i precede those in S_i-1.
end SCH
```

Figure 3.5 The Muntz-Coffman Algorithm

Set	S_5	S_4	S_3				S_2		S_1		S_0
i	1	3	5	6	7	2	8	4	10	9	11
L_i	5	4	3	3	3	3	2	2	1	1	0
H_i	5	4	3	3	3	4	2	4	1	3	0
non critical						X		X		X	
candidate set	X	X					X		X		X

Figure 3.6

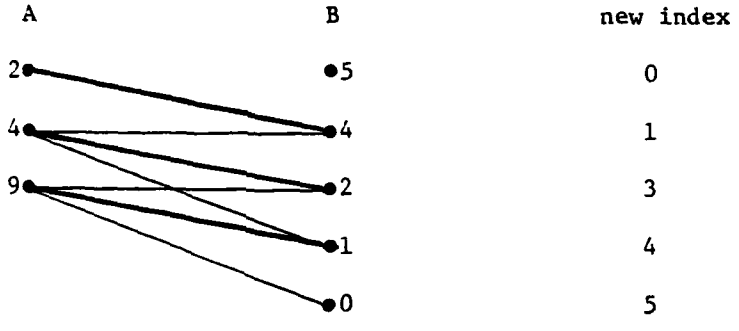


Figure 3.7 Convex Bipartite Graph.

(1) reindex the vertices in B. The new index of vertex i∈B is L-i. So, for i∈A, $s_i=L-H_i$ and $h_i=L-L_i$.
(2) use Glover's algorithm considering vertices in B in increasing order of index.

Let $S'_L, S'_{L-1}, \ldots, S'_\emptyset$ be the sets obtained by updating $S_L, S_{L-1}, \ldots, S_\emptyset$ using the maximum matching above. These updated sets satisfy the following properties:

(a) There is no job pair (i,j) such that iPj and $i \in S'_a$, $j \in S'_b$ for a $\leq b$.
(b) There is no partition of the job set J that satisfies property (a) and has a fewer number of sets of size 1 than in $S'_L, \ldots, S'_\emptyset$.

For our example, the maximum matching M is {(2,4),(4,2),(9,1)} (solid lines in Figure 3.7). So, $S'_i = S_i$, $i = \emptyset, 1, 2, 5$. $S'_3 = \{5,6,7\}$; and $S'_4 = \{3,2\}$. Properties (a) and (b) hold for our example.

The correctness of (a) follows from the indexing scheme imposed on the jobs (viz. iPj implies i<j) and the unique way in which Glover's algorithm works. Suppose that statement (a) is incorrect. Then, there exists a job pair (i,j) such that iPj and $i \in S'_a$, $j \in S'_b$ and $a \leq b$. Since no such job pair exists in $S_L, \ldots, S_\emptyset$, it must be the case that job j is noncritical and is matched to b (old indexing) in M (as only noncritical jobs can change sets and no job moves to a set of smaller index). From the definition of L and H and the knowledge that iPj, it follows that $H_i \geq L_i$, $H_i \geq L_j$, $L_i > L_j$, and $H_i > H_j$. If job i is critical, then it is the case that $L_i > H_j$. So, job j cannot possibly be matched to a set b with $a \leq b$. Hence, we may assume that i is also noncritical. Again, if i is not matched in M, then a>b as for a to be less than or equal to b, it must be the case that $L_i < H_j$. But since $L_i > L_j$, $h_i < h_j$ and i must get matched to b before j can (see Glover's algorithm). Hence, i and j must both be matched in M. i is matched to a and j to b. Hence, $a \neq b$. If a<b then vertex b is considered before a (as its new index is L-b<L-a). Since i P b, it must be the case that $H_i \geq b \geq L_i$. Also, $L_i > L_j$ implies that $h_i < h_j$. Hence, Glover's algorithm will match i to b and not j to b.

The truth of property (b) is established using the following reasoning. If $i \in B$ is such that no vertex of A is matched to i in M, then $|S'_i| = 1$. To see this, observe that S_i has exactly one critical job. Every noncritical job in S_i is a candidate for matching with $i \in B$. Thus, if no job is matched with i, then all these noncritical jobs are matched elsewhere and $|S'_i| = 1$. It is readily seen that if $i \in B$ is matched in M, then $|S'_i| > 1$. Hence, the number of sets of size 1 is minimized when a maximum matching is used.

Complexity Analysis

The steps in our parallel preemptive scheduling algorithm are:

1) Determine L_i and H_i, $1 \leq i \leq n$
2) Mark the noncritical jobs
3) Mark the candidate sets
4) Construct the convex bipartite graph

5) Find the maximum matching M
6) Reassign matched jobs to their new sets
7) Schedule jobs in the same set using McNaughtons rule.

Step 1 can be performed in $O(\log^2 n)$ time using the critical path algorithm developed by Dekel, Nassimi, and Sahni [3]. This algorithm uses $O(n^3/\log n)$ PEs. The noncritical jobs can be identified in $O(1)$ time using n PEs. The sets $S_1,\ldots,S_L$ may be obtained in $O(\log n)$ time by sorting on L_i. This requires $O(n^2)$ PEs. The candidate sets can be marked in $O(\log n)$ time and the resulting convex bipartite graph may be constructed in an additional $O(1)$ time using n^2 PEs. The matching requires $O(\log^2 n)$ time and $O(n)$ PEs while the reassigning takes $O(1)$ time. A parallel implementation of MCNaughtons rule appears in [1]. This has complexity $O(\log n)$ and uses $O(n/\log n)$ PEs.

Hence, the overall complexity of our seven step parallel algorithm for preemptive scheduling is $O(\log^2 n)$. The number of PEs used is $O(n^3/\log n)$. Since, the fastest sequential algorithm known for this problem (i.e., the Muntz-Coffman algorithm) has complexity $O(n^2)$, the EPU of our parallel algorithm is $O(n^2/(\log^2 n * n^3/\log n)) = O(1/(n\log n))$.

4. Conclusions

We have developed fast parallel algorithms for several versions of the matching problem for convex bipartite graphs. In Section 2, we explicitly considered the maximum cardinality matching problem. In Sections 3.1 and 3.2, we considered the problems of obtaining a maximum matching that minimized the maximum weight edge used as well as one that maximized the total weight of the used edges. Both these problems were discussed in connection with the scheduling problems in which they naturally arose. Finally, the maximum cardinality matching algorithm was again used to solve the preemptive scheduling problem of Section 3.3 efficiently.

This paper has further enhanced the utility of the binary tree method of Dekel and Sahni [2] for the design of parallel algorithms. It should also be pointed out that while all of our complexity analyses have assumed the availability of as many PEs as needed, our algorithms can be used when fewer PEs are available. The complexity of each algorithm will increase by no more than the shortfall in PEs. So if only half the number of PEs is available, then the time needed will at most double (except for a possible constant increase in overhead).

5. References

1. Dekel, E. and Sanni, S., "Parallel scheduling algorithm," Department of Computer Science, University of Minnesota, Technical Report 81-1.
2. Dekel, E., and Sanni, S., "Binary trees and parallel scheduling algorithms," Department of Computer Science, University of Minnesota, Technical Report 80-19.
3. Dekel, E., Nassimi, D., and Sahni, S., "Parallel matrix and graph algorithms" Proc. of Seventeenth Annual Alleerton Conf. on Communication, Control and Computing Oct 1979 pp. 27-36. (to appear in SIAM J. Computing)
4. Glover, F., "Maximum matching in a convex bipartite graph," Naval Res. Logist. Quart., 14 (1967), pp. 313-316.
5. Muntz, R. R., and Coffman, E. G., Jr., "Optimal preemtive scheduling on two-processor systems," IEEE Trans on Computer, c-18, (1969) pp. 1014-1020.
6. Nassimi, D., and Sanni S., "Bitonic sort on a mesh connected parallel computer," IEEE Trans on Computers, c-28, no. 1, January 1979, pp.2-7.
7. Preparata, F. P., "New parallel-sorting schemes," IEEE Trans. on Computers, c-27, no.7, July 1978, pp.669-673.
8. Rinnooy Kan, A. H. G., "Machine scheduling problems, classification complexity, and computation". Nignoff, The Hague, 1976.
9. Savage, C., "Parallel algorithms for graph theoretic problems," Ph.D. Thesis, University of Illinois, Urbana, August 1978.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A111 378	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Parallel Matching Algorithm for Convex Bipartite Graphs and Applications to Scheduling	5. TYPE OF REPORT & PERIOD COVERED Technical Report April 1981
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Eliezer Dekel and Sartaj Sahni	8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0650
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department University of Minnesota 136 Lind Hall, 207 Church St. SE, Mpls., MN 55455	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Department of the Navy Office of Naval Research Arlington, Virginia 22217	12. REPORT DATE April 1981
	13. NUMBER OF PAGES 25
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Parallel algorithm, convex bipartite graph, scheduling, complexity.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An efficient parallel algorithm to obtain maximum matching in convex bipartite graphs is obtained. This algorithm can be used to obtain efficient parallel algorithms for several scheduling problems. Some examples are: job scheduling with release times and deadlines; scheduling to minimize maximum completion time.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-LF-014-6601

Unclassified